高技术通讯 2012 年 第22 卷 第5 期:483 ~489

doi:10.3772/j.issn.1002-0470.2012.05.006

Xen 虚拟化环境中镜像文件的访问直接映射研究①

杨亚军②*** 高云伟*

(*中国科学院计算技术研究所系统结构重点实验室 北京 100190)

(**中国科学院研究生院 北京 100049)

摘 要 针对虚拟化环境中镜像文件模式虚拟块设备因繁琐的访问映射过程而造成的性能低下问题,提出了一种 Xen 虚拟化环境中镜像文件的访问直接映射机制,而且提出了在此机制下面向镜像文件模式的虚拟块设备到物理块设备的访问直接映射算法。根据此算法,实现了虚拟块设备和物理块设备之间的访问直接映射,该机制简化了镜像文件模式虚拟块设备 I/O 访问过程中繁琐的访问映射过程,有效地提高了镜像文件的 I/O 性能。试验表明,相对于传统的镜像文件,这样的访问直接映射可使镜像文件的 I/O 性能提高 28%。

关键词 虚拟化,镜像文件,文件块号,物理块号,直接映射

0 引 言

虚拟化技术的细粒度资源共享和高可靠的运行时隔离等特性使得其越来越成为企业数据中心或云计算平台提供商优先考虑部署的技术之一。在当前的虚拟化环境中(比如 Xen[1]),虚拟机可以通过多种模式获取存储资源服务,其中镜像文件模式由于使用灵活,并能够方便地在多个虚拟机间共享存储资源,因而得到了广泛的应用。然而,与其它模式相比,镜像文件模式因为需要繁琐的访问映射过程,因此性能较差,与其它模式的性能差距可达 50%。因此,进行提高镜像文件模式的访问性的研究很有必要。当前,研究人员在 I/O 虚拟化的性能提升方面做了很多研究,尤其是在网络 I/O 虚拟化方面。例如,文献[2-4]对 I/O 通道的优化进行了研究,文献[5-10]集中研究了对 I/O 虚拟化的硬件支持。然而,目前还没有针对镜像文件性能提升方面的研究。在 Xen 虚拟化 I/O 架构中,镜像文件模式与其它模式相比需要额外繁琐的访问映射过程,因而在 I/O 处理中需要执行更多的指令,I/O 访问性能与其它模式相比有较大差距。针对这个问题,本研究提出了 Xen 虚拟化环境中面向镜像文件的虚拟块设备到物理块设备的访问直接映射算法,并构建了新型的小而高效的映射表,进而基于此算法和映射表,实现了镜像文件模式虚拟块设备的访问直接映射,从而简化了其 I/O 处理过程,使其 I/O 性能得到了提升。与传统的镜像文件虚拟块设备相比,我们的改进可使镜像文件虚拟块设备的 I/O 性能提升 28%。

1 Xen 虚拟化环境中的 I/O 架构

在 Xen 虚拟化环境中,一个有特权的 driver domain(设备驱动域)能够直接访问物理设备,并为其它虚拟机提供 I/O 服务[11]。虚拟机将自己的 I/O 请求通过前端设备驱动发送给 driver domain 中的后端设备驱动,I/O 请求经过访问映射(将虚拟机发出的对虚拟块设备的访问映射为对物理块设备的访问)后交由物理设备驱动程序完成真正的 I/O 操作,并返回访问结果。Xen 中的虚拟块设备可以分为多种模式,如物理磁盘模式(后面简称为 physical 模式)、逻辑卷管理(LVM)模式和镜像文件模式,其中镜像文件模式在 Xen 中又可以细分为 tap-disk 子模式和 loop device 子模式[12]。图 1 是 Xen 虚拟化环境中 physical 模式的虚拟块设备的 I/O 架构图,图 2 和图 3 分别是 tap-disk 和 loop 子模式的虚拟块设备的 I/O 架构图。

① 863 计划(2009AA01Z151)和国家自然科学基金(60921002)资助项目。

② 男,1978 年生,博士生;研究方向:计算机系统结构,操作系统与虚拟化技术;联系人,E-mail: yangyajun@ncic.ac.cn

(收稿日期:2010-11-19)

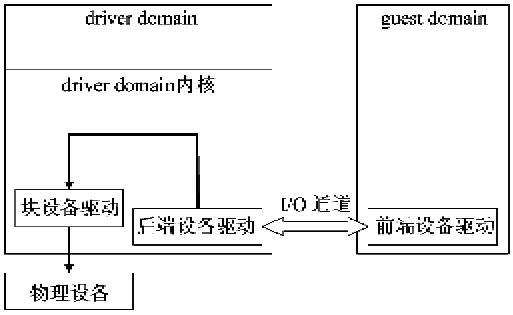

图 1 physical 模式虚拟块设备 I/O 架构

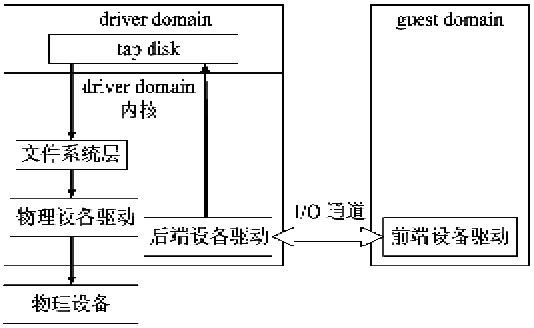

图 2 tap-disk 子模式虚拟块设备 I/O 架构

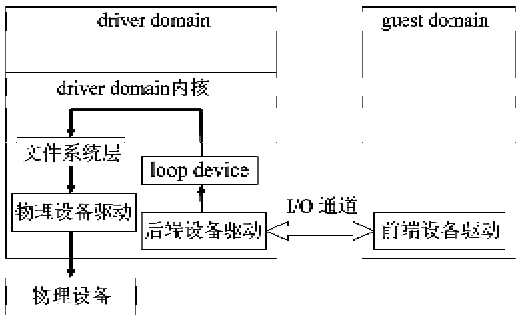

图 3 loop 子模式虚拟块设备 I/O 架构

由图 2 和图 3 可见，镜像文件模式的虚拟块设备在进行 I/O 访问时需要先通过 tap-disk 或 loop device 层将对虚拟块设备的访问映射为对镜像文件的访问，然后再通过文件系统层将对镜像文件的访问映射为对物理块设备的访问。而在 physical 模式中则没有如此繁琐的访问映射过程。额外的访问映射过程使得镜像文件在 I/O 处理中需要执行更多的指令，I/O 访问性能因此与其它模式相比有较大的差距。

2 镜像文件模式虚拟块设备的访问直接映射

由上一节可见，镜像文件模式的虚拟块设备具有额外繁琐的访问映射过程，因而性能较差。

在虚拟机的前端驱动程序中，I/O 请求会被组织起来传送给 driver domain 中的后端驱动程序。如果在前端驱动程序中能够完成虚拟块号（同时也是镜像文件中的文件块号）到物理块号的映射，则 driver domain 中的后端驱动程序可以直接把 I/O 请求交给物理设备驱动程序来完成，而无需额外的访问映射过程了，如图 4 所示。

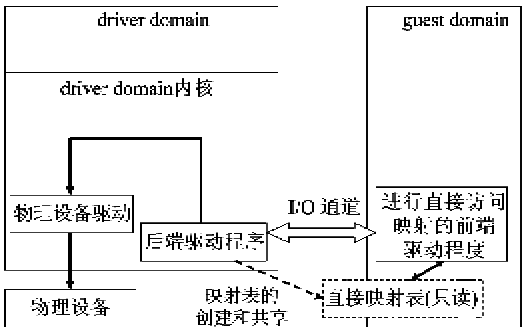

图 4 访问的直接映射机制

为了得到镜像文件的映射信息并构建映射表，后端驱动程序会在虚拟块设备的创建过程中扫描镜像文件的存储布局并构建映射表。在此映射表中，每个入口项代表镜像文件中的一个连续存储的部分，前端驱动程序可以从这些入口项中获取虚拟块号到物理块号的映射信息并完成访问的映射。这样我们就可以简化镜像文件模式虚拟块设备的 I/O 处理过程，减少需要执行的指令数量，并进而提高镜像文件的 I/O 访问性能。

为了保证其它虚拟机的数据安全，映射表对虚拟机来说是只读的。虚拟机只能从中得到分配给自己的虚拟块设备相关的映射信息。同时，后端驱动程序会对前端驱动程序中传过来的目的物理块号做安全范围检查，以保证虚拟机不会越界访问数据。

在以上访问直接映射过程中，核心工作是把虚拟块设备中的虚拟块号映射为物理块设备中的物理块号。而在镜像文件模式的虚拟块设备中，虚拟块号即为镜像文件中的文件块号。因此，在镜像文件模式下，这个核心工作就是把镜像文件的文件块号映射为镜像文件所在物理块设备中的物理块号。然而，要实现它们的直接映射是一个很大的挑战，因为镜像文件通常很大，并且在大部分情况下不是连续存储的。通常在镜像文件的存储区域中间有很多“空洞”（位于镜像文件的存储区域之间，但是存储的是其它文件的数据或者未存储数据的区域），如第 3.1 节中的图 5 所示。这些空洞增加了映射过程

的复杂性。

为了解决这个问题,文件系统比如 ext3 会为每个文件维护一个块号映射表,其中为每个文件块号都构建一个入口项,入口项中存储的是此文件块号对应的物理块号。这种机制是通用的,可以适用于各种各样的文件。然而,我们在虚拟化环境中使用的镜像文件通常都比较大,其映射表也比较大。例如,在 ext3 文件系统中,如果一个文件的大小为 10G 且块大小为 512 字节的话,那么其映射表的大小会达到 80M 字节。为了减少内存的消耗,文件系统通常只读入映射表的一部分,映射表的其它内容会在需要时才会被读入内存。这样会降低块号映射的速度,因为磁盘操作是一个相对较慢的过程。

针对上述问题,我们提出了文件块号和物理块号之间的直接映射算法和小而高效的映射表。基于新的算法和小而高效的映射表,我们就可以实现本节中提出的镜像文件虚拟块设备的访问直接映射,并进而提高其 I/O 访问性能。

3 访问直接映射算法

我们提出的直接映射算法面向连续和非连续存储的文件,同时算法所需的映射表非常小,可以全部加载入内存,从而使得映射过程中不再需要从磁盘中读入数据。因为 ext3 文件系统被广泛应用,因此我们的算法将基于 ext3 文件系统。

3.1 块号直接映射算法

非连续存储文件可以根据其存储布局被分为多个连续存储的部分,如图 5 所示:

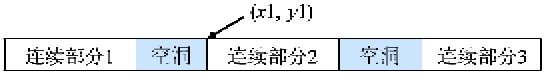

图 5 非连续存储的文件可以被分为多个连续的部分

在图 5 中,部分 2 的首文件块号为 $x1$,首物理块号为 $y1$。对于部分 2 中的某个文件块号 $x2$ 和其对应的物理块号 $y2$,$x2$ 与 $x1$ 之间的差值与 $y2$ 与 $y1$ 之间的差值未必相等。因为在 ext3 文件系统中,分配给某个文件的物理块不一定存储的是文件中的数据,它有可能存储的是块号映射表中的数据(我们称这样的块为“映射块”,而称存储文件数据的块为“数据块”),如图 6 所示。

在图 6 中,$y2$ 与 $y1$ 之间的差值要大于 $x2$ 与 $x1$ 之间的差值,因为 $y2$ 与 $y1$ 之间有一个映射块用来

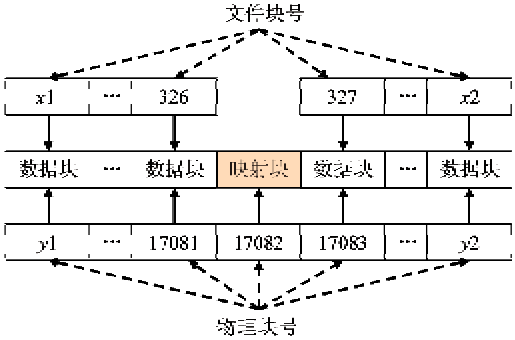

图 6 映射块对块号差值的影响示例

记录映射关系。如果能够找出某个连续存储的部分中任意两个文件块号对应的物理块号之间映射块的数量,我们就可以根据两个文件块号的差值来求得对应物理块号的差值,进而就可以根据某个连续存储部分的首文件块号和首物理块号求得此连续存储部分中任意文件块号对应的物理块号。

在 ext3 文件系统中,每个文件的 inode 会维护一个与块号映射相关的数组,它有 15 个元素,包含了一个大映射表的相关信息,这个映射表被用来将文件块号映射为物理块号。数组中的前 12 个元素包含了前 12 个文件块号(0 到 11)对应的物理块号。而第 13 到第 15 个元素分别指向了一个二级、三级和四级映射数组。第 13 个元素指向了一个二级映射数组,它包含的物理块号指向一个包含了后面最多 $b/4$ 个块号映射(b 为文件系统中块的大小,每个映射占 4 字节存储空间)的二级映射块(相应的文件块号范围:$12 - (b/4 + 11)$)。第 14 个元素指向了一个三级映射数组,它包含的物理块号指向一个包含了最多 $b/4$ 个二级数组的三级映射块(相应的文件块号范围:$((b/4) + 12) - ((b/4)2 + (b/4) + 11)$)。类似地,第 15 个元素指向一个四级映射块(相应的文件块号范围:$((b/4)2 + (b/4) + 12) - ((b/4)3 + (b/4)2 + (b/4) + 11)$)。每个映射块只有在需要的时候才会被申请,也即,映射块包含的就是紧跟其后的一定数量的数据块的块号映射关系。

因为三级映射块和四级映射块数量很少,为了简化计算,当一个连续存储的部分包含三级映射块和四级映射块时,我们将此存储部分以此三级或四级映射块为界分为两个部分,这两个部分均不包含此三级或四级映射块。这样,同一个连续部分中的两个块之间只可能包含二级映射块,可以分为三种典型的情况,如图 7、图 8 和图 9 所示(三个图中的 m、n、和 s 均小于 $b/4$,且图中的 $x1$ 和 $x2$ 均为数据

块的文件块号,而 offset 则表示 $x2$ 与 $x1$ 之间的差值)。

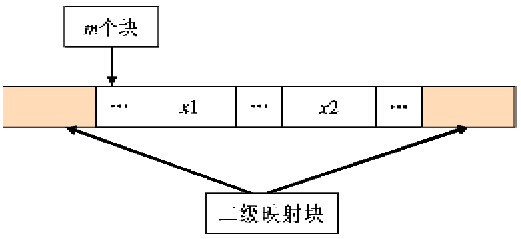

图 7 offset 小于 $b/4$,其间无映射块

在图 7 中,$x1$ 和 $x2$ 之间的 offset 小于 $b/4$,且它们之间没有二级映射块。

在图 8 中,$x1$ 和 $x2$ 之间的 offset 小于 $b/4$,但它们之间有一个二级映射块。

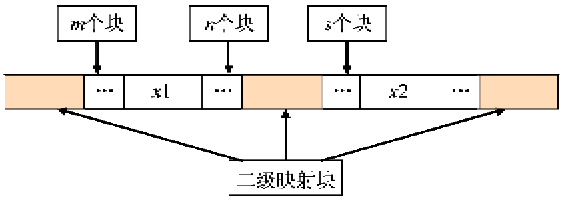

图 8 offset 小于 $b/4$,其间有映射块

在图 9 中,$x1$ 和 $x2$ 之间的 offset 大于 $b/4$,且它们之间有多个二级映射块。

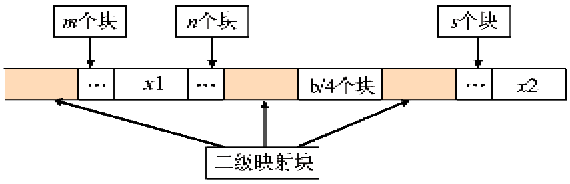

图 9 offset 大于 $b/4$,其间有多个映射块

不能直接使用(offset/($b/4$))来计算 $x1$ 和 $x2$ 之间的二级映射块数量(注意:计算符号设定如下:"/"表示相除并取整数,而"%"表示相除并取余数。)。因为即使 $x2$ 与 $x1$ 之间的 offset 小于 $b/4$,$x2$ 与 $x1$ 之间也可能存在二级映射块,如图 8 所示。因为 $x1$ 的起始位置可以是任意的:$x1$ 与前一个二级映射块之间的数据块数量 m 可以为 0 到 ($b/4-1$) 之间的任意值,$x1$ 与紧跟其后的二级映射块之间的数据块的数量 n 的取值范围也为 0 到 ($b/4-1$) (m 与 n 之间的关系为:$m+n+1=b/4$)。

然而,由上面三个图可见,$x2$ 与 $x1$ 之间二级映射块的数量等于位于 $x1$ 前面且最靠近 $x1$ 的那个二级映射块与 $x2$ 之间的二级映射块的数量。因此有如下公式:

$$num\_of\_map\_blk = (x2 - x1 + m)/(b/4) \tag{1}$$

利用式(1)可以计算 $x1$ 与 $x2$ 之间二级映射块的数量。它加上 $x1$ 与 $x2$ 之间的差值就等于 $x1$ 与 $x2$ 对应的物理块号之间的差值。因此,假设 $y1$ 为 $x1$ 对应的物理块号,$y2$ 为 $x2$ 对应的物理块号,则有:

$$y2 = y1 + x2 - x1 + (x2 - x1 + m)/(b/4) \tag{2}$$

同时可以注意到,除了前 12 个块号的映射关系是在文件的 inode 对应数组中保存之外,其它文件块号与物理块号的对应关系数据都保存在二级映射块中,且除了最后一个二级映射块之外的其它每个二级映射块均对应 ($b/4$) 个块。因此,位于 $x1$ 前面且最靠近 $x1$ 的那个二级映射块之前包含 ($12+(b/4)\times N$) 个块,如图 10 所示。

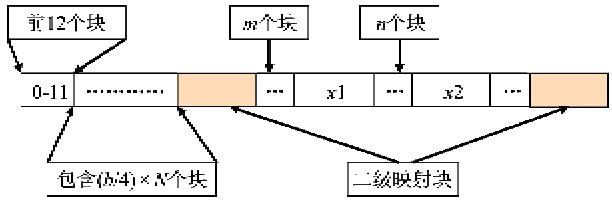

图 10 m 与 $x1$ 之间的关系

其中的 N 为 0 或正整数。因为文件块号的编号是从 0 开始,前 12 个文件块号的编号为:0 - 11,因此有公式:

$$(x1 - 11)\%(b/4) = m + 1 \tag{3}$$

把式(3)中的 1 左移即得到

$$m = (x1 - 11)\%(b/4) - 1 \tag{4}$$

把上式带入式(2)中,得到

$$y2 = y1 + x2 - x1 + (x2 - x1 + (x1 - 11)\%(b/4) - 1)/(b/4) \tag{5}$$

如果 $x1$ 与 $y1$ 分别取某个连续存储部分的首文件块号和首物理块号,则通过式(5),可以实现此连续存储部分中任意文件块号到物理块号的快速直接映射。

3.2 小而高效的映射表

文件不都是连续存储的,一个文件可能被分为多个连续存储的部分。在我们优化的镜像文件虚拟块设备 I/O 架构中,虚拟块设备在创建时,后端驱动程序会扫描镜像文件的存储布局,并创建一个映射表。在这个映射表中,每个连续存储的部分对应一

个入口项。入口项中包含此连续部分的首文件块号和首物理块号。映射表的格式如图 11 所示。

```
first_file_blk_numfirst_phy_blk_num
    {0,                 1667807 },
    {1036,              1668846 },
    {3356,              1671680 },
    {35581,             1704448 },
```

图 11　映射表结构示例

此映射表在构建后,会共享给前端驱动程序(只读共享以保证安全隔离)。有了图 11 中的信息,前端驱动就可以实现虚拟磁盘中的虚拟块号(也即镜像文件的文件块号)到物理块号的直接映射,并进而完成虚拟块设备和物理块设备之间的访问直接映射。

这个映射表很小,原因是每个入口项对应一个连续存储的部分,并且保存的信息仅仅占用 8 个字节的空间。例如,我们所使用的镜像文件(大小约 7GB)其存储布局中有 151 个连续存储的部分,对应的映射表也只有 151 个入口项,其大小不超过 2KB。

3.3　直接映射的实现代价及对镜像文件快照与克隆的影响

访问映射机制的实现包括对前后端设备驱动的修改、块号映射算法的实现、对存储布局改动的监控以及文件存储布局的扫描等部分,总的代码量约为 1400 行左右。

我们的访问直接映射机制不但能够提升镜像文件的 I/O 性能,还能够提升其快照和克隆的效率。在进行快照时,我们可以记录下上一次快照以来被写入数据的所有块号,并在下一次进行快照时仅对这些新写入数据的块进行快照即可。这显著提高了快照的效率。而在进行镜像文件克隆时,我们同样可以在克隆开始后记录下被写入数据的块号,并将要写入的数据缓冲在内存中,暂时不写入镜像文件。然后在克隆完整个镜像文件后把新写入的数据一次性更新到新的镜像文件中。这使得镜像文件可以实现在线克隆,并且不会对克隆的速度造成大的影响。

4　试验结果与分析

直接映射的实现基于 Xen 3.3 提供的虚拟化环境,对应的虚拟机操作系统为 linux,内核版本号为 2.6.18.8。试验平台为一台拥有两个双核 AMD Opteron™ Processor 275(2190MHz),4G 内存,两个 SCSI 磁盘和两个千兆网卡的服务器。

我们关心两种典型访问模式下改进后镜像文件 I/O 架构的性能:对连续存储的大文件的访问和对不连续存储的小文件的访问。在第一种模式下,使用 hdparm[13] 来测试镜像文件的峰值 I/O 性能。在第二种模式下,使用 httperf[14] 来测试不同 I/O 架构对其性能的影响,因为它能提供对大量小文件集合的访问,同时也能表现出我们进行的改进对真实服务性能的影响。

4.1　对连续存储的大文件的访问

在这种模式下,我们使用 hdparm 来测试镜像文件的峰值性能。对连续存储的大文件的访问具有很高的局部性,因此不同模式的镜像文件都具有很好的表现。试验结果显示,不同模式的镜像文件,包括优化后的镜像文件,在访问连续存储的大文件时得到了与 Host Linux 近似的峰值性能,如图 12 所示。

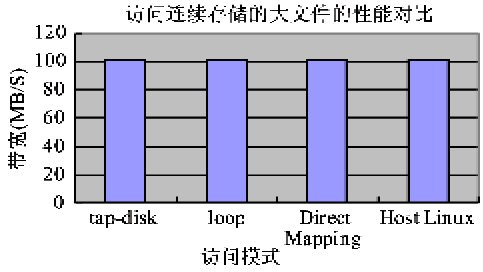

图 12　访问连续大文件的性能对比

4.2　对非连续存储的小文件集合的访问

在这种模式下,我们使用 httperf 来测试不同模式的镜像文件 I/O 架构的性能。在测试中,在一个在测试平台上运行的虚拟机中安装了 Apache 服务,并使用了一台服务器作为客户端发出 httperf 请求。测试中,客户端每次设定固定的负载(每秒钟发出固定数量的请求),然后观察其请求响应。负载从 600 开始逐渐增加到 1000。测试结果如图 13、图 14 和图 15 所示。

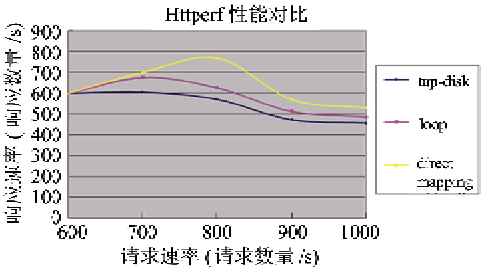

图 13　httperf 测试性能对比

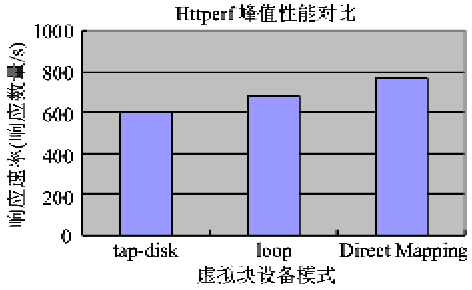

图 14 httperf 峰值性能对比

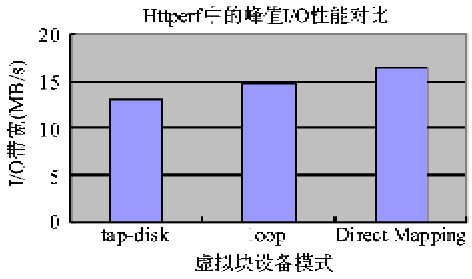

图 15 httperf 测试中峰值 I/O 性能对比

由图 13 可见,随着请求速率的提升,Web 服务性能先是逐渐提升,达到一个最高点后随着请求速率的不断增加而下降。其中 tap-disk 模式性能最差,而我们的 direct mapping 性能最好。图 14 和图 15是它们的峰值 httperf 性能和峰值 I/O 性能对比。

在图 14 和图 15 中,tap-disk 表示 tap-disk 子模式的镜像文件的性能,loop 表示 loop 子模式的镜像文件的性能,Direct Mapping 表示经过我们直接映射优化的镜像文件的性能。

试验结果表明,与 tap-disk 子模式的虚拟块设备相比,我们的改进使得 httperf 的服务性能提升达 24%。而与 loop 子模式的虚拟块设备相比,此改进使得 httperf 的服务性能有约 11% 的性能提升。

在测试中,本文也使用了 dstat[15] 来获取虚拟机的 I/O 性能,如图 15 所示。与 tap-disk 子模式的虚拟块设备相比,我们的改进使得镜像文件虚拟块设备的 I/O 性能提升达 28%。而与 loop 子模式的虚拟块设备相比,此改进使得镜像文件虚拟块设备有约 14% 的 I/O 性能提升。

本文也使用了 dbench[16] 来测试访问直接映射带来的 I/O 访问性能的提升。在测试时使用缺省的 loadfile,客户进程数为 10。图 16 是不同镜像文件模式下的测试结果对比。

如图 16 所示,与 tap-disk 子模式和 loop 子模式的虚拟块设备相比,这种的改进使得镜像文件虚拟块设备的I/O性能提升分别达27%和13%。这与

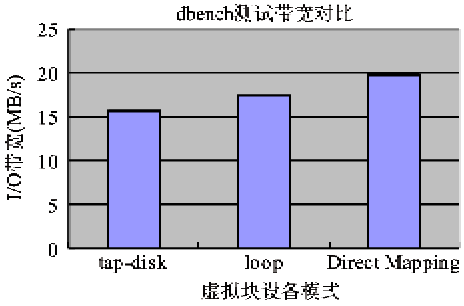

图 16 不同模式下的 dbench 测试结果对比

前面的 Httperf 的测试结果是一致的。

现在来分析本文的改进使得镜像文件性能提升的原因。本文使用 Xenoprof[17] 收集了上面测试中不同模式的镜像文件虚拟块设备对应的 I/O 操作所执行的指令数,如图 17 所示。

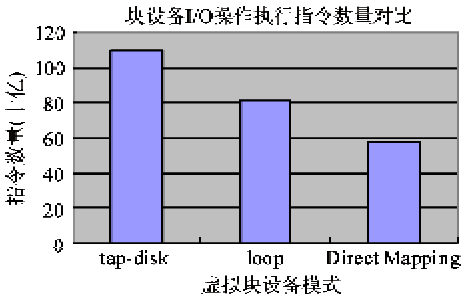

图 17 不同模式 I/O 操作执行的指令数对比

由图 17 可见,与 tap-disk 子模式相比,我们的改进使得镜像文件虚拟块设备 I/O 操作所执行的指令数减少了约 43%。而与 loop 子模式相比,我们的改进使得镜像文件虚拟块设备 I/O 操作所执行的指令数减少了约 23%。指令数的减少意味着 I/O 操作需要的时间的减少,而这意味着 I/O 性能的提升。

有时有些镜像文件的存储布局可能会发生变化。因此后端驱动程序会监控镜像文件的存储布局的改变。如果有改变发生,后端驱动程序会通知前端驱动程序暂停 I/O 处理,然后获取存储布局的改变信息并更新映射表,之后再通知前端驱动程序继续进行 I/O 处理。这样的处理过程需要暂停 I/O 处理,因此如果镜像文件的存储布局经常发生改变,则其性能会受到较大影响。在使用 dbench 进行测试后发现,在存储布局发生变化的频率较高时,使用我们的访问直接映射机制的镜像文件的 I/O 访问性能可能会下降为不到存储布局不变时的二分之一。因此未来工作将推进对其进行性能优化的研究。

5 结 论

本文针对镜像文件模式虚拟块设备的性能低下问题,提出了一种虚拟块设备到物理块设备之间的访问直接映射算法,并基于此算法实现了镜像文件模式下虚拟块设备到物理块设备的访问直接映射,提高了镜像文件的 I/O 性能。试验表明,与 tap-disk 子模式的虚拟块设备相比,这种改进使得镜像文件虚拟块设备的 I/O 性能提升达 28%。

同时,本文的优化还面临一些问题。这种优化在针对存储布局不变的镜像文件时有很好的效果,但当镜像文件的存储布局发生变化且频率较高时,I/O 性能会出现较大下降。未来的工作将对这个问题进行研究。

参考文献

[1] Barham P, Dragovic B, Fraser K, et al. Xen and the art of virtualization. In: Proceedings of the 19th ACM Symposium on Operating Systems Principles, Bolton Landing, USA, 2003. 164-177

[2] Kaushik Kumar Ram, Jose Renato Santos, Yoshio Turner, Alan L. Cox, Scott Rixner. Achieving 10 Gb/s using safe and transparent network interface virtualization. In: Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, Washington, USA, 2009. 61-70

[3] Menon A, Cox A L, Zwaenepoel A. Optimizing network virtualization in Xen. In: Proceedings of USENIX Annual Technical Conference, Boston, USA, 2006. 15-28

[4] Santos J R, Turner Y, Janakiraman G, et al. Bridging the gap between software and hardware techniques for I/O virtualization. In: Proceeding of USENIX 2008 Annual Technical Conference on Annual Technical Conference, Boston, USA, 2008. 29-42

[5] Chinni S, Hiremane R. Virtual machine device queues. http:// software. intel. com/file/1919, 2007

[6] Raj H, Schwan K. High performance and scalable I/O virtualization via self-virtualized devices. In: Proceedings of the 16th International Symposium on High Performance Distributed Computing, Monterey, USA, 2007. 179-188

[7] Willmann P, Shafer J, Carr D, et al. Concurrent direct network access for virtual machine monitors. In: Proceedings of IEEE 13th International Symposium on High Performance Computer Architecture, Phoenix, USA, 2007. 306-317

[8] PCI SIG. I/O virtualization. http://www. pcisig. com/specifications/iov/, 2010

[9] Abramson D, Jackson J, Muthrasanallur S, et al. Intel ® virtualization technology for directed I/O. http:// www. intel. com/technology/itj/2006/v10i3/2-io/1-abstract. htm, 2006

[10] IOMMU architectural specification. http://support. amd. com/us/Processor _ TechDocs/48882. pdf, 2011

[11] Fraser K, Hand S, Neugebauer R, et al. Safe hardware access with the Xen virtual machine monitor. In: Proceedings of the 1st Workshop on Operating System and Architectural Support for the on demand IT Infrastructure, 2004. 1-10

[12] Xen User's Manual. http://www. xen. org, 2003

[13] Hdparm. http://sourceforge. net/projects/ hdparm/, 2010

[14] Httperf. http://www. hpl. hp. com/research/ linux/httperf/, 1998

[15] Dstat. http://freshmeat. net/projects/dstat/, 2008

[16] Dbench. http://dbench. samba. org/web/ index. html, 2008

[17] Xenoprof. http://xenoprof. sourceforge. net, 2005

The direct access mapping for image files in xen virtualization environment

Yang Yajun*,**, Gao Yunwei*

(*Key Laboratory of Computer System and Architecture, Institute of Computing Technology, Chinese Academy of Science, Beijing 100190)

(**Graduate University of the Chinese Academy of Sciences, Beijing 100049)

Abstract

In consideration of the problem that in current virtualization environment, image files suffer from severe performance degration for their manifold access mapping proceedings, a direct access mapping mechanism for image files in the Xen virtualization environment was put forward, and based on the mechanism, a direct access mapping algorithm for access of virtual devices to physical devices was designed and implemented. This mechanism can remove the extra processing in the traditional access mapping proceedings of image files and improve their I/O performance. The experimental results show that this optimization can achieve up to 28% of performance improvement compared with the traditional image files.

Key words: virtualization, image files, file block number, physical block number, direct mapping